A Semispace-Free Cheney-Style Copying Garbage Collector
DRAFT
Update: April 28, 2011
Daniel Jeffrey Ralston

I. Cheney's Algorithm

A smashing tutorial of C.J. Cheney's algorithm[1][4] for copying garbage collection exists at [2]. Even so, a short description of Cheney's algorithm follows.

In Cheney's scheme, the heap is divided into two "semispaces" of identical size, only one of which is active at a given time. All "live" objects are copied from one semispace ("from-space") into the other ("to-space"). "From-space" is then discarded all in one piece, and becomes the "to-space" for the next collection cycle.

During a collection cycle:

All "roots" are checked. A root is any object that is immediately reachable; pointers on the stack and global variables are typically treated as roots. For each reference, or pointer, to an object in "from-space", one of two actions is taken:

- If the object has not yet been moved into "to-space", create an identical copy of the object in "to-space". Then, replace the original object with a forwarding pointer to the object's new location -- that is, the location of the copy. Then, update the original reference to point to the new copy in "to-space".

- If the object has already been moved to "to-space", follow the forward pointer and update the reference to point to the object's new location in "to-space".

The garbage collector then examines all objects in "to-space" -- that is, the objects which have already been reached and moved into the "to-space". One of the above two actions is performed on each object referenced by objects in the "to-space". Garbage collection is complete when all "to-space" references have been scanned and updated.

The algorithm doesn't need a recursive implementation, and only two pointers outside of "from-space" and "to-space" are required: a pointer to the beginning of free space in the "to-space", and a pointer to the next reference to be examined in "to-space". Forwarding pointers are used only during the garbage collection process.

II. An Improvement: Eliminating Semispaces

At the expense of two concessions, the heap doesn't need to be split into two halves; instead, the entire heap can be used.

- Each object must be the same size.
- Each object must have an extra field, a dedicated forwarding pointer, in a predictable location.

Part IV will detail how to lift the first restriction. Please, stop vomiting.

Keeping in mind the unified heap and the preceding concessions, the new algorithm differs as follows:

- As objects are reached, move them to the bottom of the same heap, just as if it were a separate "to-space", like in Cheney's algorithm. This would foolishly overwrite data which may or may not be live, except that, as objects are copied, the data at the destination and the source are swapped. (The source location isn't needed anymore, so why not?)

- Set the source location's forwarding pointer to point to the destination location.

- The destination location's forward pointer is set differently depending on its value. If it is invalid, the destination location has not been a source location during that collection cycle; update its forwarding pointer to point to the source location -- the new location of the data formerly at the destination.

- If the destination's forwarding pointer is invalid, the destination location has not been a source or destination location during that collection cycle; update its forwarding pointer to point to the source location -- the new location of the data formerly at the destination.

- If the destination location's forwarding pointer points to a 'higher' location imemory than the destination location, set its forwarding pointer to point to the source object.

- If the destination location's forwarding pointer points to a 'lower' location in memory than the destination location, don't modify its forwarding pointer; follow the forwarding pointer, and (possibly) each successive forwarding pointer, until a location is reached with a forwarding pointer to a location 'higher' in memory. This is the final destination of the data formerly at the destination location. Update this final destination's forwarding pointer to point the the source object.

Between collections, all forwarding pointers must be invalidated, so that the garbage collector will not be tricked by leftovers of the last collection. A special bit could be set in each forwarding pointer, or all forwarding pointers could simply be nullified after each cycle.

And so the need for separate semispaces is removed.

III. Problems

Though an improvement over the original copying algorithm, the presented algorithm is not without its drawbacks. The two most glaring are:

- The existence of dedicated per-object forwarding pointers.
- The requirement that each object be of the same size, so as not to corrupt forwarding pointers.

If each object, taking example from a Lisp environment, is a two-pointer "cons cell", the addition of a forwarding pointer introduces 1/3 space overhead. This is, however, still a slight improvement over the 1/2 overhead of semispaces. The requirement for a single object size is also unacceptable for a modern system, in which arrays and vectors are a necessity. The algorithm also makes no provisions for objects requiring "destructors" to be run when they are collected.

IV. Lifting The Size Restriction

It is desirable to support variably sized objects, in the interest of bringing the algorithm forward into the 1980s. There are two options for this that I will present.

The first is simply to include multiple memory management schemes in the same system. This route is definitely more work, though most professional-grade systems have multiple memory schemes, anyhow. The heap could, perhaps be split; for example, the presented algorithm would grow upward from the bottom of the heap, and some other scheme would grow downward from the top of the heap. If, as mentioned earlier, "destructors" for certain objects are desired, mark and sweep collection [3][5] might be used. (The destructors, of course, being executed during the sweep phase.)

The second possibility is to extend the presented algorithm. It's not strictly necessary for the forwarding pointers to be stored inside of the objects. As long as objects are aligned to some regular interval, each object's forwarding address can be stored in a separate array. Objects can then be of any size, so long as they are aligned. The forwarding pointers will not be overwritten by some large, irregularly-sized object. Since there is only one memory management scheme -- the presented algorithm -- to be implemented, this option is the simpler of the two.

V. Conclusion

I have presented a small improvement on C.J. Cheney's list copying/garbage collection algorithm. Specifically, the new algorithm reduces the space overhead compared to the original system presented by Cheney.

[1] C. J. Cheney. 1970. "A Nonrecursive List Compacting Algorithm". CACM. 13-11 pp677-678.
[2] http://www.cs.umd.edu/class/fall2002/cmsc631/cheney/cheney.html
[3] Paul R. Wilson. 1994. "Uniprocessor Garbage Collection Techniques". University of Texas.
[4] http://www.memorymanagement.org/glossary/c.html#cheney.collector
[5] http://www.memorymanagement.org/glossary/m.html#mark-sweep